Object-Oriented Smart Executive
FY2000 Proposal for 632-07

Task Leaders

Erann Gat, JPL, gat@jpl.nasa.gov, 818-354-4674, Principal Investigator
Marcel Schoppers, JPL, marcel@robotics.jpl.nasa.gov, 818-354-9290, Co-investigator

Product Description

A smart executive is a crucial component of current control architecture designs for autonomous spacecraft. The executive is responsible for coordinating run-time operations including plan execution, reactive recovery from contingencies, and invoking external fault recovery mechanisms. The current state of the art in autonomout executives for spacecraft is the Remote Agent Smart Executive [6, 7]. (See also the related work section on page 4.)

We propose to improve the current state of the art in three ways. First, we will **reduce the memory footprint by a factor of 3** by using a more compact implementation of the core software. This is very important because the current footprint (>10MB) is a significant obstacle to deployment in a flight environment. Second, we will **increase runtime efficiency of the system database by at least a factor of 2** by replacing the current unification database with an object database. Finally, we will **add new features to support the new JPL Mission Data System (MDS) architecture** and its associated goal-based commanding paradigm.

This is a continuing task transitioning from push to pull.

Benefits

This product provides the sequencing and system coordination capabilities for a semi-autonomous or fully autonomous spacecraft. It provides two major classes of benefits: reduced operations costs and enabling new classes of missions. Both benefits are vitally important in the new climate where missions are expected to do more with less. Both benefits were demonstrated recently on the Remote Agent Experiment on the New Millennium Space Technology 1 mission (ST1, formerly DS1) technology demonstration mission. The Remote Agent autonomy software, which includes the Smart Executive as one of three major components, controlled the ST1 spacecraft autonomously for over two days. Except for diagnosing a bug in the Remote Agent software (not unexpected since this was a technology validation) no operator intervention was needed during that time. This demonstration paves the way for a new way of doing business where a thoroughly tested industrial-strength version of the Remote Agent or its successor does most of the day-to-day operations currently performed by humans.

Our major customer is a pair of interferometry missions, New Millennium ST3 and the Space Interferometry Mission (SIM). Both of these missions are unprecedented in their level of precision, operational complexity, and workload. ST3 consists of three separate spacecraft flying in formation. SIM is only a single spacecraft, but it must configure itself with microradian and micrometer accuracy. Both missions are working observatory missions which must be constantly reconfiguring themselves in response to observation

requests. Furthermore, the observation process is an active process, requiring the spacecraft to autonomously locate and track interference fringes. It would be impossible to operate these missions using traditional open-loop sequencing.
The smart executive provides the reactive control component of an autonomous control system that **enables these missions to be operated reliably with minimal ground control staff**. It also **increases mission reliability** by hiding the underlying control details where many programming bugs occur. In a sense, the smart executive offers "structured programming" for reactive autonomous control, including recovery from unexpected contingencies, control of parallel processes, and integration with a state knowledge database. In the same way that structured programming offers increased productivity and reliability over code written with GOTO statements, the smart executive offers the same benefits over code written using direct access to operating system primitives.

Technical Approach

The problem
The central idea behind the smart executive is to capture complex control idioms inside a higher-level API. For example, consider a critical section where a read-modify-write operation must be performed without interrupts. If the underlying operating system provides only primitive operations to turn interrupts on and off one might at first be tempted to write the code in the following way (using C/Java syntax for illustration):

```
void critical_operation_1 () {
       turn_off_interrrupts();
       do_critical_operations();
       turn_on_interrupts();
}
```

But this code is flawed. If critical_operation_1 is called from within another critical section then interrupts will be turned back on prematurely. For example:

```
void critical_operation_2 () {
       turn_off_interrrupts();
       ...
       critical_operation_1();
       // Interrupts are back on here, but they should not be
       ...
       turn_on_interrupts();
}
```

The fix for this bug is to keep track of whether interrupts are on or not on entry to a critical section and restore the appropriate state on exit, e.g.:

```
boolean interrupt_status = true;  // Interrupts are on
void critical_operation_1 () {
       boolean original_interrupt_status = interrupt_status;
       if (interrupt_status) {
              turn_off_interrupts();
              interupt_status = false;
       }
       do_critical_operations();
       if (original_interrupt_status)
              turn_on_interrupts();
       }
       interrupt_status = original_interrupt_status;
}
```

This is now a fairly complicated piece of code to do a conceptually simple thing, and the situation gets rapidly worse. The critical-section idiom gets much more complicated in cases where exceptions and other unexpected contingencies can arise, requiring non-local transfers of control. Furthermore, this idiom and others like it are very common in autonomous control code, and all these idioms can interact with each other in complicated ways. It is easy to make mistakes in the application of these idioms because they are so complicated and non-intuitive. And such mistakes usually manifest themselves as intermittent catastrophic failures (e.g. a system crash) potentially resulting in loss of spacecraft. Because they are intermittent they are extremely difficult to debug. They are sensitive to subtle timing variations. A bug of this sort might not manifest itself at all on a ground testbed. Such a bug did in fact show up during the Remote Agent experiment. It never manifested itself during many thousands of hours of ground testing, but it did show up in less than 48 hours in flight, causing the RA to hang.

Our solution

We solve this problem by packaging the critical section idiom and others like it inside new control constructs. Instead of having to write the complicated code above, the user of the smart executive need only write:

```
void critical_operation_1 () {
      CRITICAL_SECTION {
            do_critical_operations();
      }
}
```

The smart executive provides similar constructs for handling unexpected contingencies (`WITH_RECOVERY_PROCEDURES`), managing parallel tasks (`TASK_NET, AND_PARALLEL, OR_PARALLEL, WITH-GUARDIAN`), implementing goal-directed behavior (`TO_ACHIEVE, ACHIEVE`) and integrating with a state knowledge database (`WITH_MAINTAINED-_PROPERTIES`). All of these constructs are integrated into a unified execution model that makes them interact with one another in intuitive ways. The underlying complexity is hidden from the user who is thus freed from the burden of insuring that all of the idioms are correct wherever they are used.

The current implementation of the smart executive has been statically analyzed using the formal analysis tool SPIN [4]. Several bugs were found and corrected, and no new bugs have been reported since. (The Remote Agent bug was due to a manually applied idiom for a feature not supported by the smart executive at the time.)

Most languages cannot be extended except by modifying their compilers. This is a difficult and expensive task, and results in a system that does not conform to the original language standard. The notable exception this rule is Lisp, which provides integrated features for extending the language entirely within the language standard. Implementing language extensions like `CRITICAL_SECTION` in Lisp is therefore much easier than in any other language. The prototype smart executive (and indeed the entire Remote Agent) was therefore implemented in Common Lisp [10].

However, Lisp presents certain challenges for production deployment. It is not as popular a language as C++ and is therefore not as well supported by industry, particularly in the flight environment. Furthermore, there is a perception of customer resistance to the use of Lisp (though in our experience the fear of customer resistance has been far more severe than any actual resistance we have found). We therefore proposed at the beginning of last year to attempt a port of the smart executive to C or C++. However, the

reviewer's response to this plan was so overwhelmingly negative that we decided to tackle the problem of producing an industrial-strength Lisp-based version instead. To do this we acquired a source license for Macintosh Common Lisp (MCL) and ported it to the flight operating system (vxWorks). MCL is uniquely suited to flight deployment because it is very small (less than 3 MB). It is also a very mature product, with a heritage extending back to 1988. (MCL was a commercial product before C++ even existed.) Because it is inevitable that significant portions of flight software will be written in C or C++ (and possibly Java) we are paying particular attention to interoperability issues. Our goal is to produce a product where the fact that there is an underlying Lisp implementation can be completely transparent to the user. We are integrating the Lisp thread scheduler with the native OS scheduler, developing CORBA interfaces, and insuring that there is an adequate foreign function interface.

Related work

Firby introduced the idea of new control constructs for reactive control in a language called RAPS [1]. In fact, RAPS was used in an early prototype of the Remote Agent, but turned out to be too cumbersome and inefficient. Simmons embedded a reactive control capability within an interprocess communications model implemented as a C library [8]. This system, called TCA, was also evaluated during early design stages of the Remote Agent. Simmons has since extended TCA to include a language called TDL [9] which is an extension to C++. A similar system called RL was developed by Lyons for industrial process control [5]. The Smart Executive is unique in having been integrated into a comprehensive system specifically tailored for controlling spacecraft, and in having been tested on an actual spacecraft.

Status and Milestones

We are currently ahead of schedule, and we have been able to expand the scope of the project somewhat. In addition to our original plan of porting MCL to vxWorks we also did a port to the Sparc architecture, so the resulting product will run on Solaris machines as well as in the flight environment. This will provide useful flexibility for projects who do initial development in Solaris. For other architectures the Smart Executive will run in all major commercial implementations of Common Lisp. There is at least one commercial implementation available for virtually every platform in common use.

FY 1999 Milestones:

- Port MCL to the flight environment (PPC/vxWorks). Completed.
- Demonstrate the prototype smart executive running in the MCL port. Completed.
- Port MCL to the Sparc/Solaris architecture. This is a task extension, currently in beta-test.
- Integrate the MCL thread scheduler with the underlying OS scheduler. Originally planned for next year. Currently in alpha test.

FY 2000 Milestones:

- Complete implementation and integration of object database
- Initial design and implementation of goal-based commanding constructs
- Finish debugging, begin documentation

FY 2001 Milestones:

- Finallize design and implementation of goal-based commanding constructs
- Demonstrate product in a realistic end-to-end mission scenario

Customer Relevance

The Smart Executive, along with other components of the Remote Agent, are currently baseline for ST3 and under consideration for SIM. Furthermore, many of the technologies in the Smart Executive are informing the design of the MDS control architecture. MDS is also relying on the MCL port as a backup for meeting several requirements for which their baseline plan is uncertain. In particular, dynamic code updates and certain algorithms requiring dynamic memory management are very problematic in C++, and it is not yet clear whether other alternatives will be a viable for flight.

Personnel

Dr. Erann Gat is the designer of ESL, the core of the Remote Agent Smart Executive, which is the current state of the art in autonomous spacecraft control. He has over twelve years of experience in the design and implentation of autonomous control architectures for mobile robots and spacecraft.

Dr. Marcel Schoppers is the originator of Universal Plans, a technology for compiling very large plans into very compact representations (in 1995 a plan encompassing 1055 states was compiled into 2000 lines of C code). He has over fifteen years of experience in autonomous control architectures and executives.

Mr. Gary Byers is the author of significant portions of Macintosh Common Lisp, including the compiler. He is one of the world's leading experts on advanced compiler technologies.

Technical References

[1] Firby, R. J., "Adaptive Execution in Dynamic Domains," Ph.D. thesis, Yale University Department of Computer Science, 1989.

[2] Gat, E., "ESL: A language for supporting robust plan execution in embedded autonomous agents," in Proc. of IEEE Aeronautics (AERO-98), Aspen, CO, IEEE Press, 1997.

[3] Gat, E. and Pell, B., "Abstract Resource Management in an Unconstrained Plan Execution System," in Proc. of IEEE Aeronautics (AERO-98), Aspen, CO, IEEE Press, 1998.

[4] Havelund, K., Lowry, M. and Penix, J. "Formal Analysis of a Spacecraft Controller Using SPIN." in Proceedings of the 4th International SPIN Workshop, November 1998, also NASA Ames Research Center Technical Report, November 1997.

[5] Lyons, D. "Representing and Analyzing action plans as networks of concurrent processes," *IEEE Transactions on Robotics and Automation*, 9(3), June 1993.

[6] Pell, B. et al., "A Remote Agent Prototype for Spacecraft Autonomy," SPIE Proceedings Volume 2810, Denver, CO, 1996.

[7] Pell, B., et al. "A Hybrid Procedural/Deductive Executive for Autonomous Spacecraft." AAAI Fall Symposium on Model-Directed Systems.

[8] Simmons, R. "An Architecture for Coordinating Planning, Sensing and Action," Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, 1990.

[9] Simmons, R. and Apfelbaum, D. "A Task Description Language for Robot Control." Submitted to IROS98.

[10] Steele, G. L. Jr., *Common Lisp, The Language*, second edition. Digital Press, 1990.

Date: Thu, 24 Jun 1999 11:45:30 -0700
From: Brad Hines <Braden.E.Hines@jpl.nasa.gov>
To: Doug.Bernard@jpl.nasa.gov, Chris.P.Jones@jpl.nasa.gov
Cc: Erann.Gat@jpl.nasa.gov
Subject: RA S/W of the year

The Remote Agent is the only implementation we know of of a number of technologies that are needed by JPL's interferometry missions, including the Space Interferometry Mission, the Space Technology 3 separated-spacecraft interferometer, and the Keck Interferometer.

Among these technologies are the ability to implement any of a number of levels of autonomy, from traditional schedule-based sequencing, through full autonomous planning and sequencing capability. This provides for a natural growth path during the development of our projects.

Interferometers have special sequencing needs relative to traditional spacecraft, in that interferometer subsystems are highly coupled and have more dependency on each other than is traditional. In a traditional spacecraft, a failed magnetometer does not affect the performance of the imaging camera. In an interferometer, however, the simultaneous operation of all subsystems is a requirement for basic instrument operation. Nonetheless, the autonomy requirement for a mission like SIM is rather severe - only one downlink is available every four days, so it is not practical to safe the spacecraft, for example, when a interferometer subsystem fails to operate as expected.

This kind of property severely taxes the ability of traditional sequencing technologies to operate these instruments. A sequencing technology that can react in real-time to anomalous behavior is important. For example, "star tracker 1 has lost lock, so I will temporarily suspend fringe tracking until the lock returns. Meanwhile, I will also suspend the generation of science data and possibly go into an engineering diagnostic mode if this has been a recurring problem." This kind of sophisticated behavior is nearly impossible to implement using traditional approaches, but is straightforward using Remote Agent.

A second feature of the Remote Agent that is invaluable during the system development, especially for testbeds, is the ability to modify sequences at runtime without rebooting. That is, a developer can work with a new sequence and fix bugs in it without having to reboot and recompile. This is critical in the development of complex sequences, where it can often take many minutes or even hours to reach a particular state. With RA's run-time reprogrammability, a bug encountered once well into a sequence can often be repaired by the developer in realtime and operation can proceed without having to repeat the long process. If some part of the process does have to be repeated, RA's automated recovery features help to restore the desired state as quickly as possible.

In summary, before the Remote Agent was available, we frankly didn't know how we were going to solve many of the autonomy issues that are inherent to doing the complex work of interferometry. RA has developed and demonstrated the technology needed to make autonomous interferometry viable, and is currently the baseline autonomy tool for the Keck Interferometer, the ST-3 interferometer, and the SIM instrument. RA control of practical interferometer operations scenarios has been demonstrated on SIM testbeds, and the technology has not lost its luster through the rigors of practical use. Within the realm of interferometry, Remote Agent is an essential and operational technology.

JET PROPULSION LABORATORY INTEROFFICE MEMORANDUM

June 25, 1999

To whom it may concern:

I am writing on behalf of the JPL Mission Data System (MDS) project in support of Erann Gat's research efforts on the object-oriented smart executive. Although MDS is not making direct use of the Smart Executive, the research results from that project other Remote Agent development efforts have been invaluable in guiding the design of the MDS control architecture.

The products of Erann's research are well suited for adoption by MDS, both as a future enhancement and as a potential option for nearer term use — something which we will understand better as details of our primary plan unfold. In particular, the Smart Executive offers a viable implementation for many problematic features that might otherwise constraint near term ambitions, such as those requiring dynamic code changes and dynamic memory management.

Many aspects of MDS have been enabled because the Remote Agent and Smart Executive paved the way. Our job will almost certainly be much more difficult if not impossible in the future without continued advanced research in these areas.

Robert D. Rasmussen
Chief Architect, Mission Data System